Appendix A

```
/*******************************************************************************

      Configuration Bits

*******************************************************************************/

      #define _CP_ON           0x000F
      #define _CP_OFF          0x3FFF
      #define _PWRTE_ON        0x3FF7
      #define _PWRTE_OFF       0x3FFF
      #define _WDT_ON          0x3FFF
      #define _WDT_OFF         0x3FFB
      #define _LP_OSC          0x3FFC
      #define _XT_OSC          0x3FFD
      #define _HS_OSC          0x3FFE
      #define _RC_OSC          0x3FFF

      #define       ___mkstr(x) #x
#ifdef PIC_PROGRAMMER
      #define      __CONFIG(x) asm("\tpsect config,class=CODE,delta=2"); \
                               asm("\tglobal\tconfig_word"); \
                               asm("config_word"); \
                               asm("\tdw "___mkstr(x))
#endif      /* End of PIC_PROGRAMMER */
#ifdef DATAIO_PROGRAMMER      /* Locate configuration at 0x0404 */
      #define      __CONFIG(x) asm("\tpsect dataio,class=CODE,delta=2"); \
                               asm("\tglobal\tconfig_word"); \
                               asm("config_word"); \
                               asm("\tdw "___mkstr(x))
#endif      /* End of DATAIO_PROGRAMMER */

/* end */

//    psect eedata,delta=2,abs,ovrld
//    org         2100h
//    db          1,2,3,4,5
```

```
/*****************************************************************************
Module Name:      nvmem.c
Number/Version:   1.00
History:
      Date              Rev         Author                  Description
      17-Dec-1998 1.00        C. Houlberg       Baseline.

Functions:
      initialize_system()           Initializes processor.
      eeprom_key_load()             Loads key from key loader and stores it in
                                    EEPROM.
      kgv_key_load()                Loads the key stored in the EEPROM into the
                                    KGV-68.
      erase_key()                   Erases key following an erase indication.
      wipe_key()                    Wipe the key from EEPROM memory.
      time_delay()                  Time delay (sets up interrupt routine).

Abstract:
    This program performs all Non-Volital Memory control functions.
      The PIC16F873 or PIC16F876 is used as the Non-Volital Memory device.
      The device signal definitions follow:
      Key Loader Data Interface Signals
1)          sense_in          Digital input         Signal activating KGV-68 for
                                                    keying
1)          fill_clk          Digital input         Non-volatile memory key load clock
1)          fill_data         Digital input         Non-volatile memory key load data
1)          var_req           Digital output        Strobe requesting key load
1)          erase             Discrete input        Analog input 2.5 Volt threshold
      Key Loader Indicator Signals
1)          kgv1_ok           Digital output        KGV1 key load accepted and OK
1)          erase_ind         Digital output        Erased key indicator
            kgv2_ok           Digital output        KGV2 key load accepted and OK
      System Interface Signals
            flight_erase      Discrete input        Analog input 22.5 Volt threshold
1)          xmtr_disable      Digital output        Transmitter disable signal
      KGV Interface Signals
1)          encr_sen_in1      Digital output        Sense signal for KGV1
1)          encr_fclk         Digital output        KGV key loading clock (1.6KHz)
1)          encr_fdata        Digital output        KGV key loading data
1)          encr_var_req      Digital input         KGV key variable request
1)          encr_ran_cp1      Digital input         KGV1 random compare OK (active
                                                    low)
            encr_sen_in2      Digital output        Sense signal for KGV2
            encr_mr           Digital output        KGV master reset
            encr_ck_ok1       Digital input         KGV1 key check OK (active low)
            encr_ck_ok2       Digital input         KGV2 key check OK (active low)
            encr_ran_cp2      Digital input         KGV2 random compare OK (active
                                                    low)

Notes:
      1)    Minimal set up I/O signals required to perform the non-volital
            memory function (single KGV-68).  The non-volatile memory function
            can therefore be implemented with a PIC16F83 microcontroller.
      2)    A PIC16F876 is used to perform the non-volital memory function for
            applications requiring two KGV-68s.  All the above signals are
            used in this implementation.
      3)    The processor is operated with a clock rate of 4MHz.  All timer
```

```
            operations must adjust prescaler and counter registers accordingly.
      4)    All timer operations are implemented with an interrupt.  Global
            variables are used to identify the timer function.
*****************************************************************************/
/*    Conditional compilation.
*/
//    #define DUAL_KGV_SYSTEM                    /* Two KGV-68s to load */
      #define MAX_KEYLOAD_ATTEMPTS  3            /* Attempts for each key copy */

/*    Programmer being used.
*/
      /* For the PIC programmer, no user defined memory section needed */
//    #define PIC_PROGRAMMER
      /* For the DataIO, the PICC command line must include -l-pdataio=0404h */
      #define DATAIO_PROGRAMMER

/*    All parameters and functions used by main() are defined in
      the following header files.
*/
#ifdef DUAL_KGV_SYSTEM
   #include <pic16876.h>
#else                                 /* Single KGV system */
   #include <pic1684.h>               /* Processor definitions */
#endif                                /* End of KGV system declaration */
   #include "config.h"                /* PIC configuration definitions */
   #include "nvmem.h"                 /* Non-Volital Memory control definitions */
   #include "size.h"                  /* Sensitive data size information */

/*    Configuration.
*/
   __CONFIG(_CP_OFF & _PWRTE_ON & _WDT_OFF & _XT_OSC);

/*    Constant definitions.
*/
   #define DEGLICH_COUNT         3          /* Consecutive active signal
                                               samples */

/*    Data storage locations.
*/
      #define PRIMARY_CW_STORAGE    0x00
      #define PRIMARY_KEY_STORAGE   (PRIMARY_CW_STORAGE + CHECK_WORD_SIZE)
      #define BACKUP_CW_STORAGE     (PRIMARY_KEY_STORAGE + KEY_SIZE)
      #define BACKUP_KEY_STORAGE    (BACKUP_CW_STORAGE + CHECK_WORD_SIZE)
      #define TOTAL_KEY_STORAGE     ((CHECK_WORD_SIZE + KEY_SIZE) << 1)

/*    Enumerations.
*/
      enum Activation
      {
            OFF,
            ON
      };

      enum Encrypter
      {
            KGV1,
            KGV2
```

```
};

enum KeySource
{
      BACKUP,
      PRIMARY
};

enum InterruptFunction
{
      START_KGV_KEY_LOAD,
      KGV_KEY_LOAD,
      END_KGV_KEY_LOAD,
      TIME_DELAY,
      FAST_FLASH,
      SLOW_FLASH
};
/* Global interrupt enable */
#define GLOBAL_ENABLE                0x80

/* (304 * 4 = 1,216 clock cycle half period => 1,645 Hz) */
/* Internal clock, low to high, prescale 1/16 */
#define KGV_KEY_LOAD_OPTION          0x03
/* 1/19 (1/16 * 1/19 = 1/304) */
#define KGV_KEY_LOAD_TMR0            (256 - 19)
/* Internal clock, low to high, prescale 1/256 */
#define TEN_MSEC_TIMER_OPTION 0x07
/* 1/39 (1/256 * 1/39 = 1/9,984) */
#define TEN_MSEC_TIMER_TMR0          (256 - 39)
/* Fast flash is 10 flashes/second slow flash is 2 flashes/second */
#define INDICATOR_FLASH_OPTION       0x07
#define INDICATOR_FLASH_TMR0  (256 - 39)
#define FAST_FLASH_COUNT             5
#define SLOW_FLASH_COUNT             25

/*
*/
Signal declarations.

#define key_loader_present     sense_in
#define kgv1_not_loaded        encr_ran_cp1
#define kgv2_not_loaded        encr_ran_cp2
#define launch_active          flight_erase
#define erase_active           erase

/*
*/
Variable declarations.

/* Source of key for key load and KGV being loaded */
unsigned char key_source = PRIMARY;
unsigned char key_destination = KGV1;
/* Key load attempts */
unsigned char kgv1_load_attempt = 0;
unsigned char kgv2_load_attempt = 0;
unsigned char kgv_load_attempted = 0;
/* Storage for timer function interrupt */
unsigned char key_addr;
unsigned char key_byte;
unsigned char shift_counter;
/* Interrupt function */
```

```
      unsigned char interrupt_function;
      /* Global timer count down */
      unsigned char timer_count;
      unsigned char fudge_count;

/*    Function definitions.
*/
      void initialize_system(unsigned char *key_present_ptr);
      void eeprom_key_load(unsigned char *key_present_ptr);
      unsigned char get_byte(void);
      void kgv_key_load(void);
      void interrupt handler(void);
      unsigned char read_eeprom(unsigned char address);
      void erase_key(void);
      void wipe_key(void);
      void time_delay(void);
      void check_eeprom(unsigned char *key_present_ptr);
      void display_load_status(void);

/*****************************************************************************
Function Name:    main()
Number/Version:
History:
   Date              Rev         Author                    Description
   17-Dec-1998 1.00        C. Houlberg       Baseline.

Input Variables:
   None.

Output Variables:
   None.

Global Variables:
   None.

Functions Referenced:
      initialize_system()           Initializes processor.
      eeprom_key_load()       Loads key from key loader and stores it in EEPROM.
      kgv_key_load()                Loads key into KGVs.  Transmitters off during load.
      erase_key()             Erases key following an erase indication.

Abstract:   Main program module to perform all Non-Volital Memory Control
      functions.
*****************************************************************************/
void main(void)
{
      /* Variable declarations */
      unsigned char key_present;

      /* Initialize the system */
      initialize_system(&key_present);

      /* Key loading */
      for(;;)
      {
            /* Check if loader is present (returns when not present) */
            if(key_loader_present)
```

```
                eeprom_key_load(&key_present);

            /* Check if key is present */
            if(key_present)
            {
                /* Load the key into the KGVs */
                if(!kgv_load_attempted)       /* Only if not previously
                                                 attempted */
                    kgv_key_load();

                /* Check for erase indication to erase key */
                if(erase_active)
                    erase_key();
            }
        }
}

/*****************************************************************************
Function Name:    initialize_system()
Number/Version:
History:
    Date              Rev         Author                    Description
    17-Dec-1998 1.00        C. Houlberg       Baseline.

Input Variables:
    key_present_ptr             Pointer to key_present flag.

Output Variables:
    None.

Global Variables:
    None.

Functions Referenced:
    time_delay()                Wait for timer count to expire.
    eeprom_read()               Get byte from EEPROM - PIC library function.

Abstract:   Initializes system for all Non-Volital Memory Control functions.
*****************************************************************************/
void initialize_system(unsigned char *key_present_ptr)
{
    /* Initialize port data direction */
    TRISA = PORT_A_DIRECTION;
    TRISB = PORT_B_DIRECTION;
#ifdef DUAL_KGV_SYSTEM
    TRISC = PORT_C_DIRECTION;
#endif

    /* Initialize port output signal levels */
    var_req = !0;                       /* Active low (not requesting load) */
    kgv1_ok = !0;                       /* Active low (KGV1 not loaded) */
    erase_ind = !0;             /* Active low (key not erased) */
    xmtr_disable = !0;                  /* Active high (transmitter disabled) */
    encr_sen_in1 = 0;           /* Active high (not loading KGV1) */
    encr_fclk = !0;             /* Active falling edge (initially high) */
    encr_fdata = !0;            /* Zero data bit */
#ifdef DUAL_KGV_SYSTEM
```

```
      kgv2_ok = !0;                        /* Active low (KGV2 not loaded) */
      encr_sen_in2 = 0;              /* Active high (not loading KGV2) */
      encr_mr = 0;                         /* Active high (not performing reset) */
#endif

      /* Initialize interrupts */
      INTCON = GLOBAL_ENABLE;        /* Global enable, mask all interrupts */

      /* Test indicators */
      kgv1_ok = 0;                         /* Indicator on */
      timer_count = 100;                   /* 1 second interval */
      time_delay();                        /* Delay for indicated count */
      kgv1_ok = !0;                        /* Indicator off */
#ifdef DUAL_KGV_SYSTEM
      kgv2_ok = 0;                         /* Indicator on */
      timer_count = 100;                   /* 1 second interval */
      time_delay();                        /* Delay for indicated count */
      kgv2_ok = !0;                        /* Indicator off */
#endif
      erase_ind = 0;                       /* Indicator on */
      timer_count = 100;                   /* 1 second interval */
      time_delay();                        /* Delay for indicated count */
      erase_ind = !0;                /* Indicator off */

      /* Scan EEPROM for presence of key */
      check_eeprom(key_present_ptr);
}

/*****************************************************************************
Function Name:    eeprom_key_load()
Number/Version:
History:
      Date              Rev         Author                    Description
      17-Dec-1998 1.00        C. Houlberg       Baseline.

Input Variables:
      key_present_ptr               Pointer to key_present flag.

Output Variables:
      None.

Global Variables:
      None.

Functions Referenced:
      time_delay()                  Wait for timer count to expire.
      get_byte()              Get byte from KYK-13 or KOI-18.
      eeprom_write()                Put byte in EEPROM - PIC library function.

Abstract:   Loads the check word and key from key loader (KYK-13 or KOI-18)
      and stores it in EEPROM.
*****************************************************************************/
void eeprom_key_load(unsigned char *key_present_ptr)
{
      /* Variable declarations */
      unsigned char byte_count, stable_count;
      unsigned char key_segment[16];       /* Temporary storage for key segment */
```

```
      /* Disable the transmitters */
      xmtr_disable = !0;

      /* Request load after an approximate 1 second delay */
      kgv1_ok = !0;                              /* Ensure indicator flash off */
#ifdef DUAL_KGV_SYSTEM
      kgv2_ok = !0;                              /* Indicator off */
#endif
      timer_count = 10;                    /* 0.1 second interval */
      time_delay();                              /* Delay for indicated count */
      var_req = 0;                               /* Active low request */

      /* Load Check Word */
      for(byte_count = 0;
            (byte_count < CHECK_WORD_SIZE) && key_loader_present; byte_count++)
      {
            /* Get one byte of fill data */
            key_segment[byte_count] = get_byte();

            /* Set request inactive (arbitrarily set done after 1st byte) */
            var_req = !0;           /* Active low request now not active */
      }

      /* Put Check Word segment into EEPROM */
      for(byte_count = 0;
            (byte_count < CHECK_WORD_SIZE) && key_loader_present; byte_count++)
      {
            eeprom_write(PRIMARY_CW_STORAGE + byte_count, key_segment[byte_count]);
            eeprom_write(BACKUP_CW_STORAGE + byte_count, key_segment[byte_count]);
      }

      /* Wait for indication that key is coming */
      if(key_loader_present)
      {
            /* Wait for disconnect (KYK-13) or fill clock (KOI-18) */
            while(key_loader_present && fill_clk);

            /* Check for disconnect */
            if(!key_loader_present) /* Loading with KYK-13 */
            {
                  /* Wait for key loader present */
                  for(stable_count = 0; (stable_count < DEGLICH_COUNT);
                        stable_count++)
                        if(!key_loader_present) stable_count = 0;

                  /* Set request inactive (arbitrarily set done after 1st byte) */
                  var_req = 0;            /* Active low request now active */
            }
      }

      /* Load Key */
      for(byte_count = 0;
            (byte_count < KEY_SIZE) && key_loader_present; byte_count++)
      {
            /* Get one byte of fill data */
            key_segment[byte_count] = get_byte();
```

```
            /* Set request inactive (arbitrarily set done after 1st byte) */
            var_req = !0;           /* Active low request now not active */
      }

      /* Put Key segment into EEPROM */
      for(byte_count = 0;
            (byte_count < KEY_SIZE) && key_loader_present; byte_count++)
      {
            eeprom_write(PRIMARY_KEY_STORAGE + byte_count, key_segment[byte_count]);
            eeprom_write(BACKUP_KEY_STORAGE + byte_count, key_segment[byte_count]);
      }

      /* Indicate key should be present (procedure completed) */
      if(key_loader_present)
      {
            *key_present_ptr = !0;

            /* Clear erase light */
            erase_ind = !0;

            /* Wait for loader to be disconnected or turned off */
            while(key_loader_present);

            /* Indicate key load should be attempted */
            kgv_load_attempted = 0;
            kgv1_load_attempt = 0;
#ifdef DUAL_KGV_SYSTEM
            kgv2_load_attempt = 0;
#endif
      }
      else
      {
            /* Check EEPROM for old key */
            check_eeprom(key_present_ptr);

            /* Display load status */
            display_load_status();
      }

      /* Enable the transmitters */
      xmtr_disable = 0;
}

/*****************************************************************************
Function Name:    kgv_key_load()
Number/Version:
History:
      Date              Rev         Author                  Description
      17-Dec-1998 1.00        C. Houlberg       Baseline.

Input Variables:
      None.

Output Variables:
      None.

Global Variables:
```

```
      None.

Functions Referenced:
      None.

Abstract:   Loads the key in EEPROM into the KGV-68s.  The transmitters are
      disabled during the key load process.  This function can be compiled
      for optimal operation with one or two KGV-68s.
*****************************************************************************/
void kgv_key_load(void)
{
      /* Disable the transmitters */
      xmtr_disable = !0;

      /* Attempt key load until maximum attempts are exceeded */
      do
      {     /* Check if KGV1 is not loaded */
            if(kgv1_not_loaded)
            {
                  /* Set KGV1 sense input active to start load */
                  encr_sen_in1 = !0;

                  /* Wait for variable request from KGV1 */
                  while(encr_var_req);          /* Active low (wait for low) */

                  /* Attempt a key load */
                  /* Set up for start of key load interrupt */
                  interrupt_function = START_KGV_KEY_LOAD;
                  key_destination = KGV1;

                  /* Initialize timer and enable interrupt */
                  TMR0 = KGV_KEY_LOAD_TMR0;
                  OPTION = KGV_KEY_LOAD_OPTION;
                  INTCON = GLOBAL_ENABLE;
                  T0IE = 1;

                  /* Wait for key load to complete */
                  while(T0IE == 1);

                  /* Set KGV1 sense input inactive */
                  encr_sen_in1 = 0;

                  /* Count key load attempts */
                  ++kgv1_load_attempt;
            }

#ifdef DUAL_KGV_SYSTEM
            /* Check if KGV2 is not loaded */
            if(kgv2_not_loaded)
            {
                  /* Set KGV2 sense input active to start load */
                  encr_sen_in2 = !0;

                  /* Wait for variable request from KGV2 */
                  while(encr_var_req);          /* Active low (wait for high) */

                  /* Attempt a key load */
```

```
                /* Set up for start of key load interrupt */
                interrupt_function = START_KGV_KEY_LOAD;
                key_destination = KGV2;

                /* Initialize timer and enable interrupt */
                TMR0 = KGV_KEY_LOAD_TMR0;
                OPTION = KGV_KEY_LOAD_OPTION;
                INTCON = GLOBAL_ENABLE;
                T0IE = 1;

                /* Wait for key load to complete */
                while(T0IE == 1);

                /* Set KGV2 sense input inactive */
                encr_sen_in2 = 0;

                /* Count key load attempts */
                ++kgv2_load_attempt;
            }
#endif
            /* Next try other key source */
            key_source = !key_source;

            /* Delay to allow the KGV-68 time to process segment */
            timer_count = 100;
            time_delay();
        }
        /* Alternate between the primary key and the backup key */
        while((kgv1_not_loaded
                && (kgv1_load_attempt < (MAX_KEYLOAD_ATTEMPTS << 1)))
#ifdef DUAL_KGV_SYSTEM
                || (kgv2_not_loaded
                && (kgv2_load_attempt < (MAX_KEYLOAD_ATTEMPTS << 1)))
#endif
        );

        /* Enable the transmitters */
        xmtr_disable = 0;

        /* Display indication if properly loaded */
        display_load_status();

        /* Indicate KGV load attempted */
        kgv_load_attempted = !0;
}

/*****************************************************************************
Function Name:    handler()
Number/Version:
History:
        Date              Rev         Author                    Description
        17-Dec-1998 1.00        C. Houlberg       Baseline.

Input Variables:
        None.

Output Variables:
```

```
      None.

Global Variables:
      None.

Functions Referenced:
      eeprom_read()                 Get byte from EEPROM - PIC library function.

Abstract:   Loads the key in EEPROM into the KGV-68.
*****************************************************************************/
void interrupt handler(void)
{
      /* Variable declarations */
      unsigned char temp;

      /* Clear TMR0 flag */
      T0IF = 0;

      /* Determine function of interrupt */
      switch(interrupt_function)
      {
            case(START_KGV_KEY_LOAD):
                  /* Continue timer interrupt key load function */
                  TMR0 = KGV_KEY_LOAD_TMR0;
                  OPTION = KGV_KEY_LOAD_OPTION;
                  T0IE = 1;

                  /* Initialize transfer variables */
                  encr_fclk = !0;               /* Active falling edge */
                  if(key_source == PRIMARY)    /* Point to start of primary key */
                        key_addr = PRIMARY_CW_STORAGE;
                  else  /* Point to start of backup key */
                        key_addr = BACKUP_CW_STORAGE;

                  key_byte = read_eeprom(key_addr++); /* Get first byte */
                  if(key_byte & 0x80)                       /* Determine state of MSB */
                        encr_fdata = 1;                     /* Output bit */
                  else
                        encr_fdata = 0;          * Output bit */
                  key_byte = key_byte << 1;     /* Shift for next bit transfer */
                  shift_counter = 1;                  /* Indicate first bit output */

               /* New interrupt function (continue key load on next interrupt) */
                  interrupt_function = KGV_KEY_LOAD;
                  break;
            case(KGV_KEY_LOAD):
                  /* Continue timer interrupt key load function */
                  TMR0 = KGV_KEY_LOAD_TMR0;
                  OPTION = KGV_KEY_LOAD_OPTION;
                  T0IE = 1;

                  /* Transition clock */
                  temp = encr_fclk;
                  encr_fclk = !temp;

                  /* Shift out new data bit on falling edge of encr_fclk */
                  if(encr_fclk)
```

```
        {
              /* Check if starting a new byte */
              if(!shift_counter)
                    key_byte = read_eeprom(key_addr++);

              /* Check value of MSB and output to KGV */
              if(key_byte & 0x80)
                    encr_fdata = 1;
              else
                    encr_fdata = 0;

              /* Shift data and updata shift counter modulo 8 */
              key_byte = key_byte << 1;     /* Shift for next bit
                                                transfer */
              shift_counter = ++shift_counter & 0x07;
        }

        /* Check for new timer function */
        if(((key_addr == (PRIMARY_CW_STORAGE + TOTAL_KEY_STORAGE))
              || (key_addr == (BACKUP_CW_STORAGE + TOTAL_KEY_STORAGE)))
              && !shift_counter && !encr_fclk)
              interrupt_function = END_KGV_KEY_LOAD;
        break;
case(END_KGV_KEY_LOAD):
        /* Mask timer interrupt (key load function completed) */
        T0IE = 0;

        /* Toggle clock high */
        encr_fclk = !0;
        break;
case(TIME_DELAY):
        /* Reload TMR0 */
        TMR0 = TEN_MSEC_TIMER_TMR0;
        OPTION = TEN_MSEC_TIMER_OPTION;
        if(timer_count)
        {
              --timer_count;
              T0IE = 1;
        }
        else
        {
              T0IE = 0;
        }
        break;
case(FAST_FLASH):
case(SLOW_FLASH):
        /* Reload TMR0 */
        TMR0 = INDICATOR_FLASH_TMR0;
        OPTION = INDICATOR_FLASH_OPTION;
        if(timer_count)
        {
              --timer_count;
        }
        else
        {
              /* Re-establish count */
              if(interrupt_function == FAST_FLASH)
```

```
                        timer_count = FAST_FLASH_COUNT;
                    else
                        timer_count = SLOW_FLASH_COUNT;
                    /* Toggle indicator */
                    temp = kgv1_ok;
                    kgv1_ok = !temp;
#ifdef DUAL_KGV_SYSTEM
                    temp = kgv2_ok;
                    kgv2_ok = !temp;
#endif
                }
                T0IE = 1;
                break;
            default:
                break;
        }
}

/*****************************************************************************
Function Name:    read_eeprom()
Number/Version:
History:
    Date              Rev         Author                  Description
    17-Dec-1998 1.00        C. Houlberg       Baseline.

Input Variables:
    unsigned char address   EEPROM data address location.

Output Variables:
    unsigned char read_eeprom()         Data from EEPROM.

Global Variables:
    None.

Functions Referenced:
    None.

Abstract:   EEPROM read routine ONLY for interrupt handler routine.
*****************************************************************************/
unsigned char read_eeprom(unsigned char address)
{
    EEADR = address;
    RD = 1;
    return EEDATA;
}

/*****************************************************************************
Function Name:    erase_key()
Number/Version:
History:
    Date              Rev         Author                  Description
    17-Dec-1998 1.00        C. Houlberg       Baseline.

Input Variables:
    None.

Output Variables:
```

```
      None.

Global Variables:
      None.

Functions Referenced:
      wipe_key()                    Perform a wipe operation to erase the key.

Abstract:   Erases the key stored in EEPROM following an erase indication
      from the key loader.
*****************************************************************************/
void erase_key(void)
{
      /* Variable declarations */
      unsigned char stable_count;
      unsigned char delete_key = !0;

      /* Debounce the erase indication signal */
      for(stable_count = 0; stable_count < DEGLICH_COUNT; stable_count++)
            if(!erase_active) delete_key = 0;

      /* If indicated, delete the key */
      if(delete_key)
      {
            /* Wipe the key from EEPROM memory */
            wipe_key();

            /* Set erase light when key is erased */
            erase_ind = 0;

            /* Maintain KGV-68 load status */
            display_load_status();
      }
}

/*****************************************************************************
Function Name:    wipe_key()
Number/Version:
History:
      Date              Rev      Author                  Description
      17-Dec-1998 1.00        C. Houlberg       Baseline.

Input Variables:
      None.

Output Variables:
      None.

Global Variables:
      None.

Functions Referenced:
      eeprom_read()                 Get byte from EEPROM - PIC library function.

Abstract:   Performs a "wipe" operation to erase the key stored in EEPROM.
*****************************************************************************/
void wipe_key(void)
```

```
{
      /* Variable declarations */
      unsigned char key_erased = 0;
      unsigned char erase_pass;
      unsigned char byte_count;
      unsigned char data_byte[] = {0xaa, 0x55, 0x46, 0xff, 0x00};

      while(!key_erased)
      {
            /* Erase EEPROM key storage memory (5 passes) */
            for(erase_pass = 0; erase_pass < 5; erase_pass++)
            {
                  /* Perform one erasure pass */
                  for(byte_count = 0; byte_count < TOTAL_KEY_STORAGE; byte_count++)
                        eeprom_write(PRIMARY_CW_STORAGE + byte_count,
                              data_byte[erase_pass]);
            }

            /* Read EEPROM to verify erasure */
            key_erased = !0;                              /* Assume key was erased */
            for(byte_count = 0; byte_count < TOTAL_KEY_STORAGE; byte_count++)
                  if(eeprom_read(PRIMARY_CW_STORAGE + byte_count))
                        key_erased = 0;
      }
}

/*****************************************************************************
Function Name:    time_delay()
Number/Version:
History:
      Date              Rev         Author                    Description
      17-Dec-1998 1.00        C. Houlberg       Baseline.

Input Variables:
      None.

Output Variables:
      None.

Global Variables:
      None.

Functions Referenced:
      None.

Abstract:   Set up timer counter and waits until interrupts are completed.
*****************************************************************************/
void time_delay(void)
{
      interrupt_function = TIME_DELAY;/* Set up interrupt */
      TMR0 = TEN_MSEC_TIMER_TMR0;          /* Initialize timer */
      OPTION = TEN_MSEC_TIMER_OPTION;
      INTCON = GLOBAL_ENABLE;              /* Ensure interrupts are enabled */
      T0IE = 1;                                  /* Enable timer interrupt */
      while(T0IE == 1);                    /* Wait for delay to complete */
}
```

```
/*****************************************************************************
Function Name:    get_byte()
Number/Version:
History:
      Date              Rev         Author                    Description
      17-Dec-1998 1.00        C. Houlberg       Baseline.

Input Variables:
      None.

Output Variables:
      unsigned char byte

Global Variables:
      None.

Functions Referenced:
      None.

Abstract:   Gets a byte from the KYE-13 or KOI-18.
*****************************************************************************/
unsigned char get_byte(void)
{
      /* Variable declarations */
      unsigned char bit_count, stable_count;
      unsigned char data_byte;

      /* Get one byte of fill data (clocked in on falling edge) */
      for(bit_count = 0;
            (bit_count < 8) && key_loader_present; bit_count++)
      {
            /* Wait for clock to go low */
            for(stable_count = 0;
                  (stable_count < DEGLICH_COUNT) && key_loader_present;
                  stable_count++)
                  if(fill_clk) stable_count = 0;

            /* Put data bit into data byte (MSB first) */
            if(key_loader_present)
                  data_byte = (data_byte << 1) + fill_data;

            /* Wait for clock to go high */
            for(stable_count = 0;
                  (stable_count < DEGLICH_COUNT) && key_loader_present;
                  stable_count++)
                  if(!fill_clk) stable_count = 0;
      }

      /* Return data byte */
      return data_byte;
}

/*****************************************************************************
Function Name:    check_eeprom()
Number/Version:
History:
      Date              Rev         Author                    Description
```

```
      17-Dec-1998 1.00        C. Houlberg       Baseline.

Input Variables:
      unsigned char *key_present_ptr

Output Variables:
      None.

Global Variables:
      None.

Functions Referenced:
      None.

Abstract:   Scans EEPROM for possible presence of key.
*****************************************************************************/
void check_eeprom(unsigned char *key_present_ptr)
{
      /* Variable declarations */
      unsigned char i;
      unsigned char no_data = !0;          /* Assume no data in EEPROM */
      unsigned char bad_data = 0;          /* Primary and backup data matches */
      unsigned char stored_value;

      for(i = 0; i < BACKUP_CW_STORAGE; i++)
      {
            stored_value = eeprom_read(PRIMARY_CW_STORAGE + i);
            if(stored_value && (stored_value != 0xff))
                  no_data = 0;                  /* Have data in EEPROM */
            if(stored_value != eeprom_read(BACKUP_CW_STORAGE + i))
                  bad_data = !0;                /* Mismatch => bad key data */
      }
      if(no_data || bad_data)
      {
            *key_present_ptr = 0;    /* No good key data */

            /* Flash kgv_ok to indicate the key is no good (10 flashes/sec) */
            interrupt_function = FAST_FLASH;    /* Set up interrupt */
            TMR0 = INDICATOR_FLASH_TMR0;        /* Initialize timer */
            OPTION = INDICATOR_FLASH_OPTION;
            timer_count = FAST_FLASH_COUNT;     /* 0.05 second interval */
            T0IE = 1;
            kgv1_ok = 0;                              /* Indicator on */
#ifdef DUAL_KGV_SYSTEM
            kgv2_ok = 0;                              /* Indicator on */
#endif
      }
      else
      {
            /* Have a key */
            *key_present_ptr = !0;
      }
}

/*****************************************************************************
Function Name:    display_load_status()
Number/Version:
```

```
History:
      Date                Rev         Author                    Description
      17-Dec-1998 1.00        C. Houlberg       Baseline.

Input Variables:
      None.

Output Variables:
      None.

Global Variables:
      None.

Functions Referenced:
      None.

Abstract:   Indicates if KGV-68 was properly loaded.
*****************************************************************************/
void display_load_status(void)
{
   if(kgv1_not_loaded)
   {
         /* Flash kgv_ok to indicate the load is no good */
         interrupt_function = SLOW_FLASH;     /* Set up interrupt */
         timer_count = SLOW_FLASH_COUNT;
         TMR0 = INDICATOR_FLASH_TMR0;         /* Initialize timer */
         OPTION = INDICATOR_FLASH_OPTION;
         INTCON = GLOBAL_ENABLE;            /* Ensure interrupts are enabled */
         T0IE = 1;
         kgv1_ok = 0;                                 /* Indicator on (toggle) */
   }
   else
   {
         kgv1_ok = 0;                   /* Properly loaded (indicator on) */
   }
#ifdef DUAL_KGV_SYSTEM
   if(kgv2_not_loaded)
   {
         /* Flash kgv_ok to indicate the load is no good */
         interrupt_function = SLOW_FLASH;     /* Set up interrupt */
         timer_count = SLOW_FLASH_COUNT;
         TMR0 = INDICATOR_FLASH_TMR0;         /* Initialize timer */
         OPTION = INDICATOR_FLASH_OPTION;
         INTCON = GLOBAL_ENABLE;            /* Ensure interrupts are enabled */
         T0IE = 1;
         kgv2_ok = 0;                                 /* Indicator on (toggle) */
   }
   else
   {
         kgv2_ok = 0;                   /* Properly loader (indicator on) */
   }
#endif
}

/* end */
```

```
/*****************************************************************************
Module Name:       nvmem.h
Number/Version:    1.00
History:
      Date              Rev         Author            Description
      17-Dec-1998       1.00        C. Houlberg       Baseline.

Abstract:    Project definitions.
*****************************************************************************/

/*    Constant definitions.
*/
      /* Key Loader Data Interface Signals */
      #define sense_in         RA0          /* Signal activating KGV-68 for keying */
      #define fill_clk         RA1          /* Non-volatile memory key load clock */
      #define fill_data        RA2          /* Non-volatile memory key load data */
      #define var_req          RA3          /* Strobe requesting key load */
      #define erase            RA4          /* Analog input         2.5 Volt threshold */
      /* Key Loader Indicator Signals */
      #define kgv1_ok          RB0          /* KGV1 key load accepted and OK */
      #define erase_ind        RB1          /* Erased key indicator */
      #define kgv2_ok          RC0          /* KGV2 key load accepted and OK */
      /* System Interface Signals */
      #define flight_erase     RA5          /* Analog input         22.5 Volt threshold */
      #define xmtr_disable     RB2          /* Transmitter disable signal */
      /* KGV Interface Signals */
      #define encr_sen_in1     RB3          /* Sense signal for KGV1 */
      #define encr_fclk        RB4          /* KGV key loading clock */
      #define encr_fdata       RB5          /* KGV key loading data */
      #define encr_var_req     RB6          /* KGV key variable request strobe */
      #define encr_ran_cp1     RB7          /* KGV1 random compare OK (active low) */
      #define encr_sen_in2     RC3          /* Sense signal for KGV2 */
      #define encr_mr          RC4          /* KGV master reset */
      #define encr_ck_ok1      RC5          /* KGV1 key check OK (active low) */
      #define encr_ck_ok2      RC6          /* KGV2 key check OK (active low) */
      #define encr_ran_cp2     RC7          /* KGV2 random compare OK (active low) */

      /* Port A and B data direction (1/0 => Input/Output) */
#ifdef DUAL_KGV_SYSTEM
      #define PORT_A_DIRECTION            0x37
      #define PORT_B_DIRECTION            0xc0
      #define PORT_C_DIRECTION            0xe6
#else /* Single KGV system */
      #define PORT_A_DIRECTION            0x17
      #define PORT_B_DIRECTION            0xc0
#endif

/* end */
```

```
/*
 *    Header file for the Microchip
 *    PIC 16CR83 chip
 *    PIC 16F83 chip
 *    PIC 16C84 chip
 *    PIC 16F84 chip
 *    PIC 16CR84 chip
 *    Midrange Microcontrollers
 */

static volatile unsigned char RTCC        @ 0x01;
static volatile unsigned char TMR0        @ 0x01;
static volatile unsigned char PCL         @ 0x02;
static volatile unsigned char STATUS      @ 0x03;
static          unsigned char FSR         @ 0x04;
static volatile unsigned char PORTA       @ 0x05;
static volatile unsigned char PORTB       @ 0x06;
static volatile unsigned char EEDATA      @ 0x08;
static volatile unsigned char EEADR       @ 0x09;
static          unsigned char PCLATH      @ 0x0A;
static volatile unsigned char INTCON      @ 0x0B;

static          unsigned char bank1 OPTION      @ 0x81;
static volatile unsigned char bank1 TRISA       @ 0x85;
static volatile unsigned char bank1 TRISB       @ 0x86;
static volatile unsigned char bank1 EECON1      @ 0x88;
static volatile unsigned char bank1 EECON2      @ 0x89;

/*    STATUS bits */
static volatile bit     RP0   @ (unsigned)&STATUS*8+5;
static volatile bit     TO    @ (unsigned)&STATUS*8+4;
static volatile bit     PD    @ (unsigned)&STATUS*8+3;
static volatile bit     ZERO  @ (unsigned)&STATUS*8+2;
static volatile bit     DC    @ (unsigned)&STATUS*8+1;
static volatile bit     CARRY @ (unsigned)&STATUS*8+0;

/*      PORTA bits      */
static volatile bit      RA4     @ (unsigned)&PORTA*8+4;
static volatile bit      RA3     @ (unsigned)&PORTA*8+3;
static volatile bit      RA2     @ (unsigned)&PORTA*8+2;
static volatile bit      RA1     @ (unsigned)&PORTA*8+1;
static volatile bit      RA0     @ (unsigned)&PORTA*8+0;

/*      PORTB bits      */
static volatile bit      RB7     @ (unsigned)&PORTB*8+7;
static volatile bit      RB6     @ (unsigned)&PORTB*8+6;
static volatile bit      RB5     @ (unsigned)&PORTB*8+5;
static volatile bit      RB4     @ (unsigned)&PORTB*8+4;
static volatile bit      RB3     @ (unsigned)&PORTB*8+3;
static volatile bit      RB2     @ (unsigned)&PORTB*8+2;
static volatile bit      RB1     @ (unsigned)&PORTB*8+1;
static volatile bit      RB0     @ (unsigned)&PORTB*8+0;
static volatile bit      INT     @ (unsigned)&PORTB*8+0;

/*    INTCON bits */
static volatile bit     GIE   @ (unsigned)&INTCON*8+7;
static volatile bit     EEIE  @ (unsigned)&INTCON*8+6;
```

```
static volatile bit     T0IE  @ (unsigned)&INTCON*8+5;
static volatile bit     INTE  @ (unsigned)&INTCON*8+4;
static volatile bit     RBIE  @ (unsigned)&INTCON*8+3;
static volatile bit     T0IF  @ (unsigned)&INTCON*8+2;
static volatile bit     INTF  @ (unsigned)&INTCON*8+1;
static volatile bit     RBIF  @ (unsigned)&INTCON*8+0;

/*    OPTION bits */
static bank1 bit  RBPU        @ (unsigned)&OPTION*8+7;
static bank1 bit  INTEDG      @ (unsigned)&OPTION*8+6;
static bank1 bit  T0CS        @ (unsigned)&OPTION*8+5;
static bank1 bit  T0SE        @ (unsigned)&OPTION*8+4;
static bank1 bit  PSA         @ (unsigned)&OPTION*8+3;
static bank1 bit  PS2         @ (unsigned)&OPTION*8+2;
static bank1 bit  PS1         @ (unsigned)&OPTION*8+1;
static bank1 bit  PS0         @ (unsigned)&OPTION*8+0;

/*     TRISA bits      */
static volatile bank1 bit       TRISA4  @ (unsigned)&TRISA*8+4;
static volatile bank1 bit       TRISA3  @ (unsigned)&TRISA*8+3;
static volatile bank1 bit       TRISA2  @ (unsigned)&TRISA*8+2;
static volatile bank1 bit       TRISA1  @ (unsigned)&TRISA*8+1;
static volatile bank1 bit       TRISA0  @ (unsigned)&TRISA*8+0;

/*     TRISB bits      */
static volatile bank1 bit       TRISB7  @ (unsigned)&TRISB*8+7;
static volatile bank1 bit       TRISB6  @ (unsigned)&TRISB*8+6;
static volatile bank1 bit       TRISB5  @ (unsigned)&TRISB*8+5;
static volatile bank1 bit       TRISB4  @ (unsigned)&TRISB*8+4;
static volatile bank1 bit       TRISB3  @ (unsigned)&TRISB*8+3;
static volatile bank1 bit       TRISB2  @ (unsigned)&TRISB*8+2;
static volatile bank1 bit       TRISB1  @ (unsigned)&TRISB*8+1;
static volatile bank1 bit       TRISB0  @ (unsigned)&TRISB*8+0;

/*   EECON1 bits */
static volatile bank1 bit     EEIF  @ (unsigned)&EECON1*8+4;
static volatile bank1 bit     WRERR @ (unsigned)&EECON1*8+3;
static volatile bank1 bit     WREN  @ (unsigned)&EECON1*8+2;
static volatile bank1 bit     WR    @ (unsigned)&EECON1*8+1;
static volatile bank1 bit     RD    @ (unsigned)&EECON1*8+0;


/* macro versions of EEPROM write and read */
#define     EEPROM_WRITE(addr, value)
while(WR)continue;EEADR=(addr);EEDATA=(value);GIE=0;WREN=1;\
                            EECON2=0x55;EECON2=0xAA;WR=1;WREN=0
#define     EEPROM_READ(addr) ((EEADR=(addr)),(RD=1),EEDATA)

/* library function versions */

extern void eeprom_write(unsigned char addr, unsigned char value);
extern unsigned char eeprom_read(unsigned char addr);


#define CONFIG_ADDR     0x2007
#define FOSC0           0x01
#define FOSC1           0x02
```

```
#define WDTE            0x04
#define PWRTE           0x08

/* code protection */
#if defined (_16C84)
#define CP              0x10
#endif

#if defined (_16CR83) || defined(_16CR84)
#define DP              0x80
#define CP              0x3F70
#endif

#if defined (_16F83) || defined(_16F84)
#define CP              0x3FF0
#endif

#define UNPROTECT CP
#define PROTECT         0x0000
```